Implementacja środowiska inżynierskiego na przykładzie pakietu CPDev

Dariusz Rzońca, Jan Sadolewski, Andrzej Stec, Zbigniew Świder, Bartosz Trybus, Leszek Trybus
Politechnika Rzeszowska, Katedra Informatyki i Automatyki, ul. W. Pola 2, 35-959 Rzeszów

Streszczenie: Norma IEC 61131-3 definiuje pięć języków programowania sterowników przemysłowych. Norma ta jest powszechnie stosowana, wiele środowisk inżynierskich jest z nią całkowicie, bądź częściowo zgodnych. W literaturze opisano kilka akademickich rozwiązań, jednak zazwyczaj implementują one jedynie wybrane elementy normy (np. tylko jeden lub dwa języki). Komercyjne środowiska inżynierskie zwykle obsługują wszystkie języki, ale ich dokumentacja skupia się na korzystaniu ze środowiska, natomiast rzadko ujawniane są szczegóły dotyczące wewnętrznej architektury i implementacji. W artykule przedstawiono takie rozwiązania dla pakietu inżynierskiego CPDev. Architektura bazująca na maszynie wirtualnej sprawia, że środowisko jest przenośne, co ułatwia wdrożenie na różnych platformach sprzętowych. W artykule przedstawiono kilka wdrożeń przemysłowych środowiska CPDev

Słowa kluczowe: środowisko inżynierskie, IEC 61131-3, język pośredni, maszyna wirtualna, translator, HMI, komunikacja, sterownik przemysłowy

1. Wprowadzenie

Środowiska inżynierskie implementujące języki normy IEC 61131-3 (przyjętej w Polsce jako PN-EN 61131-3) są obecnie powszechnie stosowane w przemyśle do tworzenia programów sterowania [1, 2]. Standard IEC 61131-3 [3] definiuje pięć języków programowania: dwa tekstowe IL, ST, dwa graficzne LD, FBD i mieszany język SFC. Języki IL, LD i SFC są przeznaczone zwłaszcza do zastosowań w procesach produkcyjnych opartych na sterownikach PLC [4], natomiast ST, FBD, jak też SFC, są stosowane w obszarach wykorzystujących sterowniki PAC i systemy DCS.

Kompilatory języków IEC 61131-3 są podstawowymi składnikami środowisk inżynierskich. Kompilator, na podstawie kodu źródłowego programu, generuje kod wykonywalny. Zwykle stosowane są trzy podejścia do kompilacji programów sterowania. Pierwsze polega na bezpośrednim tłumaczeniu programu sterowania z języków IEC 61131-3 do wykonywalnego kodu maszynowego procesora. Ponieważ zmiana procesora (CPU) wymaga nowego kompilatora, takie rozwiązanie jest zasadniczo dedykowane dla konkretnej platformy sprzętowej.

Kolejne podejście jest dwuetapowe. W pierwszym kroku programy sterowania są tłumaczone z języków IEC 61131-3 na kod źródłowy w języku pośrednim C lub C++, który następnie, kolejnym narzędziem, jest kompilowany do wykonywalnego kodu maszynowego. Rozwiązanie to może być przeniesione na różne platformy sprzętowe, jeśli tylko posiadamy dla nich kompilator języka C/C++.

Trzecie z podejść wymaga opracowania maszyny wirtualnej (VM) interpretującej pewien pośredni język, do którego kompilowane są programy sterowania. Takie podejście zostało wprowadzone na przykład w środowisku ISaGRAF [5] z językiem pośrednim TIC (ang. *Target Independent Code*). Obecnie STRATON [6] (niezależny dostawca) również bazuje na takiej metodzie, jak również wiele z prototypowych rozwiązań akademickich także bazuje na maszynach wirtualnych. Jednak maszyna wirtualna wprowadza dodatkowy narzut czasowy, co powoduje, że tego typu podeście jest zwykle mniej wydajne.

Prace nad środowiskiem inżynierskim zgodnym z normą IEC 61131-3 autorzy tego artykułu rozpoczęli ponad 10 lat temu [7], po rozmowach z producentami automatyki przemysłowej. W pewnym stopniu była to kontynuacja wcześniejszych prac nad sterownikami wielofunkcyjnymi [8]. Opracowane środowisko, nazwane CPDev (ang. *Control Program Developer*) kompilowało programy z języka ST do kodu VMASM (ang. *Virtual Machine Assembler*) interpretowanego przez maszynę wirtualną, początkowo na ośmiobitowych mikrokontrolerach MCS 51 i AVR. Z założenia środowisko wspierało samodzielną adaptację maszyny wirtualnej dla docelowego sprzętu przez inżynierów i studentów, zarówno dla zastosowań w przemyśle, jak i w dydaktyce. W kolejnych latach w pakiecie wprowadzono rozszerzenia dla 32- i 64-bitowych procesorów, uzupełniono je o translatory języków graficznych, narzędzie do projektowania interfejsów HMI i wieloprojektowe środowisko uruchomieniowe (początkowo zaimplementowane w FPGA [9]). Dwa rozszerzenia wsparły badania nad podejściem MDD (ang. *Model Driven Development*) [10] i testami jednostkowymi [11]. Do tej pory środowisko zostało wdrożone przez kilku producentów urządzeń automatyki w kraju i za granicą.

Autor korespondujący:
Dariusz Rzońca, drzonca@prz-rzeszow.pl

Artykuł recenzowany
nadesłany 16.10.2019 r., przyjęty do druku 27.11.2019 r.

W artykule przedstawiono rozwiązania podstawowych problemów, które można napotkać podczas rozwoju oprogramowania zgodnego z normą IEC 61131-3, w tym zarówno pakietu inżynierskiego, jak też środowiska uruchomieniowego. W szczególności pokazano rozwiązania zapewniające:
- przenośność kompilatora i środowiska uruchomieniowego dla różnych procesorów i zastosowań wymagających dedykowanych sterowników PLC,
- integrację elementów sprzętowych ze środowiskiem uruchomieniowym,
- mechanizm projektowania bibliotek HMI obejmujących obiekty graficzne wyświetlane na różnych urządzeniach,
- wsparcie dla wymiany danych za pomocą wielu protokołów komunikacyjnych.

Artykuł ma następującą strukturę. W kolejnym rozdziale przedstawiono przegląd literatury dotyczący podstawowych aspektów związanych ze środowiskami inżynierskimi. Rozdział 3 prezentuje możliwości CPDev. Język pośredni VMASM, parametryzowany kompilator i przykładowy program sterowania opisano w rozdziale 4. Kolejny rozdział wyjaśnia architekturę maszyny wirtualnej oraz powiązanie z funkcjami niskopoziomowymi. Rozdział 6 opisuje translatory języków graficznych i przedstawia dwuwarstwową strukturę obsługi HMI. Rozdział 7 pokazuje dane niezbędne do powiązania zmiennych z interfejsami komunikacyjnymi i przedstawia typowe wdrożenia.

2. Przegląd literatury

Koncepcja maszyny wirtualnej, jako abstrakcyjnego procesora dedykowanego języka, pierwotnie została wdrożona jako p-maszyna dla języka programowania Pascal. Współcześnie zastosowano ją w języku Java jako JVM (ang. *Java Virtual Machine*) [12] i na platformie .NET jako CLR (ang. *Common Language Runtime*) [13]. Dostępna jest także otwarta implementacja JVM w języku Verilog, przeznaczona dla FPGA [14].

W odniesieniu do systemów sterowania pierwsze rozwiązania bazujące na maszynach wirtualnych (poza komercyjnym ISaGRAF) opisano w [15, 16], stosując IL jako język pośredni. Stosunkowo prosta architektura [15] umożliwiła implementację na ośmiobitowym mikroprocesorze C8051. Ostatnio także CLR, a więc maszyna wirtualna .NET, została przystosowana do aplikacji IEC 61131-3 [17].

Powstały również narzędzia o przeznaczeniu dydaktycznym, ograniczone do wybranych elementów normy IEC 61131-3, np. UniSim [18] lub Whimori CDU [19]. Z nowszych przykładów WEB PLC Simulator [20] pozwala na implementację modelu fabryki i sterowania w języku ST. HT-PLC [21] bazujący na Arduino z modułem Ethernet jest programowalny w LD.

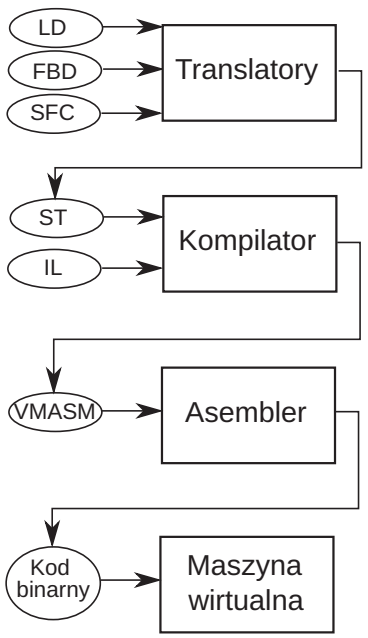
Mimo licznych aplikacji, szczegóły dotyczące kompilatorów języków IEC 61131-3 są ujawniane jedynie sporadycznie. Możliwości standardowej składni SSA (ang. *Static Single Assignment* – jednokrotne statyczne przypisanie) i zasady stosowanej tu translacji kierowanej składnią opisano m.in. w [22].

Translatory diagramów LD na języki IL lub ST zazwyczaj bazują na grafach AOV (ang. *Activity on Vertex*) [23, 24]. Translator wbudowany w CPDev również korzysta z grafu AOV, z którego generowany jest program w języku ST. Grafy EDFG (ang. *Enhanced Data Flow Graphs*) [25] pozwalają na implementowanie programów napisanych w IL, LD i SFC w FPGA.

Rozważając interfejsy HMI należy wspomnieć o zasadach graficznego projektowania aplikacji sterowania [26]. Istnieją także przemysłowe standardy w tym obszarze, jak na przykład niemiecki standard VDI/VDE 3699 [27].

3. Środowisko CPDev

Pierwotna wersja CPDev, wspierająca początkowo jedynie język ST [7], została rozbudowana o obsługę pozostałych języków normy IEC 61131-3, projektowanie interfejsów HMI, generator dokumentacji oraz moduły o zastosowaniu badawczym. Na rysunku 1 przedstawiono kolejne etapy przetwarzania programów sterowania. Programy tekstowe (języki ST, IL) są kompilowane do mnemoników pośredniego języka VMASM. Diagramy LD, FBD i SFC są tłumaczone na ST, a następnie kompilowane. Język ST jest więc bazowym językiem środowiska. Wykonywalny kod binarny jest generowany przez asembler z mnemoników VMASM i ładowany do maszyny wirtualnej. Źródła maszyny wirtualnej napisano w języku C, co ułatwia przenośność na różne platformy sprzętowe.



Rys. 1. Przetwarzanie programów sterowania w środowisku CPDev
Fig. 1. Processing of control programs in the CPDev environment

Z punktu widzenia użytkownika podstawowe funkcjonalności CPDev są podobne do innych środowisk inżynierskich. Pewnym wyróżnikiem jest tu mechanizm automatycznego wyznaczania połączeń między blokami w edytorach FBD i LD [28], który bazuje na algorytmie A* [29]. Również edytor ekranów HMI korzysta z niezależnych od sprzętu komponentów zapisanych w bibliotekach. Kompilator generuje dodatkowo plik XML z zestawem danych o rozmieszczeniu zmiennych globalnych, co jest wymagane dla interfejsów komunikacyjnych i modułów wejść/wyjść.

Na podstawie modeli w diagramach SysML [30] możliwe jest generowanie szablonów programów i zadań komunikacyjnych [10]. Testy jednostkowe i tablicowe mogą korzystać z IEC 61131-3 lub dedykowanego języka [11]. Moduły CPDev mają wbudowane interfejsy XML, co pozwala na zwinne i wahadłowe podeście inżynierii oprogramowania [31].

Poza maszyną wirtualną wykonującą zadania sterowania, środowisko uruchomieniowe może zawierać moduł wykonawczy HMI obsługujący interfejs graficzny. Maszyna wirtualna i moduł HMI działają równolegle i niezależnie, wymieniając dane przez zmienne globalne.

4. Język pośredni i kompilator

Definicja języka pośredniego VMASM obejmuje wszystkie typy danych, instrukcje i składnie. Wieloplatformowe implementacje wymagają jednak parametryzowanego kompilatora.

4.1. Instrukcje i składnia

VMASM obsługuje wszystkie typy danych zdefiniowane w normie IEC 61131-3 poza typem `WSTRING`. Z uwagi na występowanie problemów podczas mieszania typów łańcuchowych jednobajtowych z dwubajtowymi zdecydowano pozostawić w implementacji tylko jeden z nich (tablice konwersji oparte na języku i stronie kodowej, w których mają być te teksty umieszczone, ponadto występowanie znaków modyfikujących litery znacznie utrudnia implementację takich konwersji). Z uwagi na małą przestrzeń adresową w 16-bitowej wersji maszyny wirtualnej zdecydowano pozostać tylko przy typie `STRING`. W niektórych wdrożeniach CPDev, gdzie znaki międzynarodowe są niezbędne, typ `STRING` jest interpretowany jako wielobajtowy i jest równoważny typowi WSTRING.

Typ `BOOL` zajmuje 1 bajt, `INT` – 2 bajty, `REAL` – 4 bajty, `DATE_AND_TIME` – 8 bajtów (struktura z polami); podobnie dla pozostałych typów. Przez parametryzację kompilatora liczba typów wymagana w konkretnym zastosowaniu może być ograniczona.

Instrukcje VMASM składają się z funkcji i procedur. Procedury kontrolują przepływ realizacji programu, wywołują podprogramy, obsługują kopiowanie obszarów pamięci itd.

Instrukcje VMASM mają następującą składnię:

```
[:etykieta] instrukcja [operand1]
[,operand2] [,operand3] ...
```

Etykieta jest opcjonalna, instrukcja określa jednocześnie liczbę operandów. Operandami mogą być zmienne, etykiety i bezpośrednie wartości. Symbol specjalny `?` (znak zapytania) oznacza etykietę lub zmienną tymczasową z prefiksem `?LR?`. Elementy struktur są wybierane operatorem kropka (.).

W przypadku języka IL, wynik wykonywania rozkazu przechowywany jest w rejestrze CR (ang. *Current Result*), a następnie zawartość CR jest zapisywana w zmiennej. VMASM łączy te dwa kroki w jedną instrukcję, gdzie `operand1` wskazuje lokalizację wyniku (operacja pamięć-pamięć).

Kompilator i maszyna wirtualna, zaprojektowane dla wieloplatformowych implementacji, muszą brać pod uwagę rozmiar adresów docelowego procesora, bądź też nie wymagają wszystkich typów danych zdefiniowanych w normie IEC 61131-3. By to uwzględnić, kompilator ST na VMASM jest parametryzowany za pomocą pliku XML LCF (ang. *Library Configuration File*) określającego zasoby sprzętowe, typy danych, instrukcje i konwersje. Tak więc różne wersje, przeznaczone na docelowe platformy sprzętowe lub do określonych zastosowań, mogą być tworzone z jednego ogólnego kompilatora i powiązanej z nim ogólnej maszyny wirtualnej.

4.2. Przykład kompilacji

Plik XML z kodem źródłowym programu jest kompilowany przez skaner i parser do pliku z mnemonikami VMASM. Za pomocą osobnego zestawu słów kluczowych skaner ST przetwarza także język IL, zastępując rejestr CR tworzonymi *ad hoc* tymczasowymi zmiennymi [32]. Parser wyprowadza drzewo składni abstrakcyjnej według schematu z góry na dół, stosując translację kierowaną składnią [22]. Podstawowe komponenty skanera i parsera są deklarowane jako klasy w języku C#. Listy stosowane są we wszystkich fazach kompilacji.

Przykład fragmentu programu w języku ST i odpowiadającego mu tłumaczenia VMASM dla generatora pojedynczego impulsu na wyjściu `OUT` po czasie `T#5s` od aktywacji zboczem narastającym na wejściu `IN` pokazano na rysunku 2. `Blok R_TRIG1` wykrywa zbocze narastające, natomiast przerzutnik `RS1` i timer `TON1` ze sprzężeniem zwrotnym generują impuls.

Parser początkowo zapisuje rezultat do zmiennych tymczasowych (SSA zapisuje do wirtualnych rejestrów). Następnie, aby ograniczyć liczbę takich zmiennych, kompilator przechodzi przez fazę optymalizacji, w której zastępuje nadmiarowe zmienne tymczasowe zmiennymi użytkownika (globalnymi lub lokalnymi) oraz nakłada na siebie adresy tych zmiennych tymczasowych, które nie są używane do obliczeń w tym samym czasie.

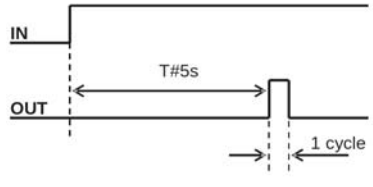
5. Maszyna wirtualna

Mnemoniki generowane przez kompilator podlegają konwersji do kodu binarnego dla maszyny wirtualnej. Maszyna jest zaprojektowana w architekturze harwardzkiej (rozdzielone przestrzenie adresowe pamięci programu i danych). W pierwszej fazie implementacji tworzony jest jej interfejs dla docelowej platformy sprzętowej.

5.1. Architektura i działanie

Podstawowe składniki maszyny wirtualnej, takie jak pamięci programu i danych, stosy programu i danych, moduł wykonywania instrukcji i interfejs dla docelowej platformy, pokazano na rysunku 3. Moduł wykonywania instrukcji pobiera rozkazy z pamięci programu, przetwarza je, pobiera operandy z pamięci danych lub programu (zmienne lub stałe) i zapisuje wyniki w pamięci danych.

```
R_TRIG1(CLK:=IN);
RS1(S:=R_TRIG1.Q, R1:=TON1.Q);
TON1(IN:=RS1.Q1, PT:=T#5s);
OUT:=TON1.Q;
```



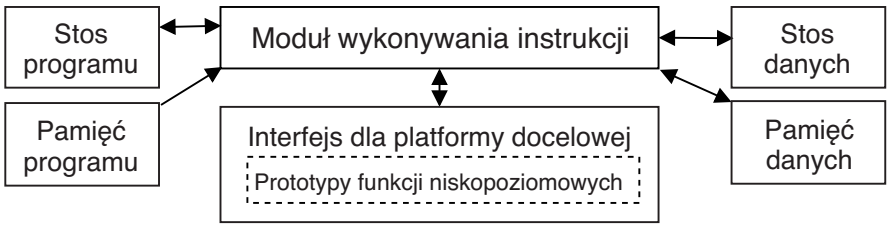

```
MEMCP R_TRIG1.CLK, IN, #0100
CALB R_TRIG1, :?IEC_61131.R_TRIG?CODE

MEMCP RS1.S, R_TRIG1.Q, #0100
MEMCP RS1.R1, TON1.Q, #0100
CALB RS1, :?IEC_61131.RS?CODE

MEMCP TON1.IN, RS1.Q1, #0100
MCD TON1.PT, #04, #88130000
CALB TON1, :?IEC_61131.TON?CODE

MEMCP OUT, TON1.Q, #0100
```

Rys. 2. Program w ST i translacja na VMASM dla generatora impulsu
Fig. 2. ST program and VMASM translation for the pulse generator



Rys. 3. Ogólna architektura maszyny wirtualnej
Fig. 3. General architecture of the virtual machine

Interfejs dla docelowej platformy składa się z niskopoziomowych funkcji zależnych od sprzętu i systemu operacyjnego, jak aktualizacja wejść i wyjść, zarządzanie czasem cyklu, alokacja pamięci itp. Ogólna część maszyny wirtualnej jest napisana w standardowym języku C, podobnie jak większość tego typu maszyn wirtualnych i może być bezpośrednio kompilowana na typowe procesory. Maszyna wirtualna działa więc jak interpreter wykonując kolejne linie kodu.

Pojedyncza instrukcja programu pobiera operandy w zależności od specyfikacji zdefiniowanej w pliku LCF i wykonuje stosowne obliczenia. W przypadku funkcji rezultat umieszczany jest pod odpowiednim adresem.

5.2. Interfejs dla platformy docelowej

Ogólna część maszyny wirtualnej zawiera zestaw funkcji do obsługi komponentów procesora w kolejnych fazach działania maszyny, rozpoczynając od załadowania programu z danego projektu. Są to prototypy niskopoziomowych funkcji (początkowo puste), niezależne od CPU i zastosowanych rozwiązań sprzętowych. Cztery przykłady takich prototypów pokazano poniżej.

- `VMP_PreRunConfiguration` – inicjalizacja sprzętu, zapis początkowego stanu (czas, parametry),
- `VMP_PreCycle` – aktualizacja zmiennych z zewnętrznych wejść przed rozpoczęciem obliczeń, zapis początkowego stanu zegara systemowego,
- `VMP_PostCycle` – aktualizacja zewnętrznych wyjść rezultatem obliczeń z bieżącego cyklu, wykonanie procedur testowych lub komunikacyjnych,
- `VMP_CurrentTime` – zwraca bieżącą wartość zegara systemowego do zmiennej typu TIME.

Kod niskopoziomowych funkcji jest przygotowywany w C/C++ podczas wdrożenia na docelową platformę i konsolidowany z ogólną częścią maszyny wirtualnej. Po tej fazie implementacja maszyny wirtualnej dla docelowej platformy sprzętowej jest gotowa do uruchomienia.

6. Translatory i HMI

Jak wspomniano w rozdziale 3, programy utworzone w językach graficznych są początkowo tłumaczone do ST, a następnie kompilowane. Dzięki dwuwarstwowej strukturze oprogramowania dla paneli HMI, narzędzie do projektowania interfejsów graficznych może współpracować z różnymi urządzeniami.

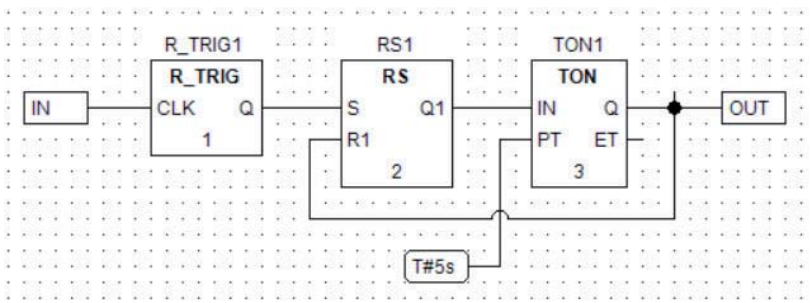
6.1. Translatory języków FBD i LD

Wspólną cechą edytorów diagramów LD i FBD jest mechanizm automatycznego wyznaczania połączeń za pomocą algorytmu A* [29]. Translatory zastępują wewnętrzne połączenia pomocniczymi zmiennymi lokalnymi niezbędnymi, by wyświetlić bieżącą wartość na połączeniu w trybie on-line.

Wersja FBD przykładowego fragmentu programu, wraz z jego translacją na ST, jest pokazana na rysunku 4a. Pomocnicza zmienna `out_R_TRIG1_Q` i dalsze z przedrostkiem `out_` są tworzone automatycznie. Poza tymi zmiennymi tłumaczenie nie różni się zbytnio od oryginalnego kodu ST (rys. 2). Rysunek 4b przedstawia diagram LD odpowiadający FBD. Sygnał `OUT` na wejściu `RS1` zapewnia sprzężenie zwrotne. Nie ma ograniczeń dla złożoności szczebla, o ile tylko zachowano reguły języka LD.

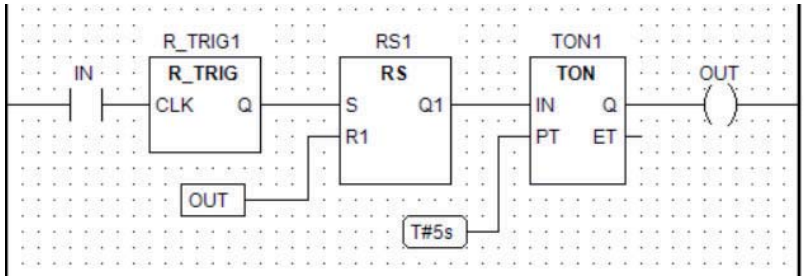
Zmodyfikowany algorytm A* stosowany w edytorach łączy podejście konwencjonalne z heurystycznym. Podczas wyzna-

(a)




```
R_TRIG1(CLK:=IN,Q=>out_R_TRIG1_Q);
RS1(S:=out_R_TRIG1_Q,R1:=out_TON1_Q,Q1=>out_RS1_Q1);
TON1(IN:=out_RS1_Q1,PT:=T#5s,Q=>out_TON1_Q);
OUT := out_TON1_Q;
```

(b)




```
out_contact_IN_80_90 := IN;
R_TRIG1(CLK:=out_contact_IN_80_90,Q=>out_R_TRIG1_Q);
RS1(S:=out_R_TRIG1_Q,R1:=OUT,Q1=>out_RS1_Q1);
TON1(IN:=out_RS1_Q1,PT:=T#5s,Q=>out_TON1_Q);
OUT := out_TON1_Q;
```

Rys. 4. Diagramy i translacja generatora impulsu: (a) FBD, (b) LD
Fig. 4. Diagrams and translation of the pulse generator: (a) FBD, (b) LD

czania połączenia obliczany jest jego koszt, biorący pod uwagę liczbę przecięć z innymi połączeniami i liczbę zmian kierunków [28], aby możliwie zmniejszyć długość połączeń, zachowując jednocześnie przejrzystość schematu.

6.2. Translator języka SFC

Diagramy SFC przechowywane są w pamięci jako drzewa. Drzewo może składać się z kroków, tranzycji, skoków, wyboru sekwencji i sekwencji współbieżnych. Pierwsze trzy elementy są podstawowe i tworzą dwa dalsze. Akcje przywiązane są do kroków, a typy akcji określane są kwalifikatorem zgodnie z normą IEC 61131-3 [3].

Translacja z SFC na ST korzysta z dwóch bloków funkcjonalnych z biblioteki systemowej, `SFC_STEP` oraz `SFC_ACTION_CONTROL`. Blok `SFC_STEP` wyznacza aktywność poszczególnych kroków na podstawie flagi *next-step-flag* powiązanej z wejściem. Blok ma dwa wyjścia, `X` i `T`, oznaczające aktywność i czas, który minął od aktywacji (dla tranzycji i akcji). Blok `STEP_ACTION_CONTROL` aktywuje akcję powiązaną z krokiem w zależności od kwalifikatora. Translator łączy wyjście `X` z `SFC_STEP` z wejściem `SFC_ACTION_CONTROL` odpowiadającym użytemu kwalifikatorowi. Wykonywanie kodu akcji uruchamiane jest wyjściem `A`.

6.3. Interfejs HMI

Graficzne panele operatorskie, zazwyczaj z ekranem dotykowym, stosowane są w systemach wykorzystujących sterowniki PLC i PAC. Panele takie niekiedy zintegrowane są ze sterownikami, jak np. w przypadku urządzeń pochodzących z firm Beckhoff, Unitronix czy Horner.

Obsługa paneli HMI została zaimplementowana w środowisku CPDev w ramach narzędzia CPVis obejmującego edytor graficzny dedykowany do tworzenia ekranów z predefiniowanych komponentów. Zmienne globalne, bądź stałe, przypisywane są jako atrybuty komponentów. Dane projektu zapisywane są w pliku XML i eksportowane do kodu binarnego wykorzystywanego w `środowisku` uruchomieniowym docelowego sterownika.

Aby zaimplementować oprogramowanie HMI na docelowym urządzeniu, inżynier powinien przygotować zestaw procedur rysujących podstawowe elementy graficzne (np. linia, koło, tekst), wykorzystując mechanizmy i biblioteki specyficzne dla urządzenia, jak np. RamTex, GLCD, GDI+ lub inne. Na podstawowych elementach bazuje domyślna biblioteka CPVis ze złożonymi obiektami graficznymi jak *BarGraph*, *ProcessValue*, *TouchableBox*, *Gauge* itp. Dzięki temu utworzone obiekty są niezależne od sprzętu i mogą być wyświetlone na różnych urządzeniach.

7. Przegląd komunikacji i implementacji

Podczas wdrażania środowiska na docelowej platformie należy także opracować mechanizmy komunikacji i konfiguracji wejść/wyjść. Aby to umożliwić, kompilator, poza kodem binarnym, tworzy plik XML (rozszerzenie DCP) z mnemonikami VMASM (dla debugowania) i podstawowymi danymi na temat zmiennych globalnych. Na podstawie tych danych możliwe jest powiązanie zmiennych z zewnętrznymi interfejsami.

7.1. Przykład narzędzia konfiguratora

Konfigurator zapewnia trzy funkcjonalności:
- ustalenie parametrów interfejsu komunikacyjnego,
- powiązanie zmiennych globalnych z interfejsami,
- przesłanie kodu binarnego, parametrów komunikacyjnych i tablicy powiązań do sterownika.

Typowe parametry interfejsu komunikacyjnego to prędkość transmisji, numer urządzenia itd. (dla łącza szeregowego) czy adres IP (dla łącza Ethernet).

Aby wyjaśnić, jak może wyglądać prosty konfigurator, rozważmy system, gdzie sterownik komunikuje się z trzema zdal-

Tabela zadań komunikacyjnych

Nazwy

Lp.	Nr urządz.	Funkcja	Adr. zdalny	Liczba	Nazwa	Konwersja
1	1	FC3-odczyt rej.	4003	1	IN	16->8 bit
2	2	FC16-zapis rej.	4205	1	OUT	8->16 bit

Rys. 5. Konfigurator komunikacji
Fig. 5. Communication configurer

nymi modułami wejścia/wyjścia [33] w protokole Modbus RTU (pierwsze przemysłowe wdrożenie CPDev [7]).

Załóżmy, że zmienna `IN` z przykładu jest ustawiana pierwszym wejściem z modułu wejść binarnych (urządzenie podrzędne nr 1, adres Modbus 4003). Podobnie zmienna `OUT` jest zapisywana do modułu wyjść binarnych (urządzenie podrzędne 2, adres Modbus 4205). Zasadnicza część okna konfiguracyjnego zawierająca definicje zadań komunikacyjnych pokazana jest na rysunku 5. Pierwszy wiersz z funkcją FC3 Modbus odczytuje jeden rejestr z urządzenia nr 1 spod adresu 4003 i zapisuje wartość do zmiennej `IN`. Drugi wiersz z funkcją FC16 zapisuje zmienną `OUT` pod adres 4205 urządzenia nr 2. Ponieważ zmienna typu `BOOL` zajmuje jeden bajt, wykonywane są konwersje 8- 16-bit.

Obecna implementacja maszyny wirtualnej na komputer PC (softcontroller) zawiera wbudowany podstawowy serwer opracowany na podstawie publicznie dostępnego LightOPC [34].

7.2. Przykładowe wdrożenia

Jak wspomniano, środowisko inżynierskie może być przeniesione na docelową platformę po opracowaniu:
- interfejsu dla platformy docelowej,
- narzędzia konfigurującego komunikację i wejścia/wyjścia,
- modułu uruchomieniowego umożliwiającego przesłanie kodu binarnego i danych konfiguracyjnych do sterownika.

Podsystemy automatyki okrętowej, jak sterowanie napędami, kursem, zarządzanie poborem energii i inne, należące do systemu Mega-Guard z firmy Praxis [35] są najistotniejszym wdrożeniem CPDev. Każdy podsystem składa się z procesora sterującego, jednostki bądź jednostek wejścia/wyjścia i panelu dotykowego TFT, bazujących na CPU ARM. Procesor komunikuje się z modułami I/O przez CAN, a z TFT przez Ethernet. Wartości zmiennych procesowych pokazywane są na ekranie TFT.

Sterownik StTr-PLC, składnik zintegrowanej platformy komunikacyjnej GSM/GPRS z firmy Nauka i Technika [36], nadzoruje i steruje rozproszonymi instalacjami. Sterownik zawiera procesor ARM-Cortex CPU, dotykowy panel TFT, wejścia/wyjścia i moduł komunikacyjny. Poza komunikacją GSM/GPRS, sterownik StTr-PLC implementuje Wi-Fi oraz ZigBee. Dzięki komunikacji bezprzewodowej urządzenia z serii StTr są także używane w zastosowaniach mobilnych, jak monitoring miejskiej komunikacji autobusowej.

Inne przykładowe wdrożenia obejmują jednostki RTU z firmy iGrid [37] do automatyki podstacji energetycznych (protokoły IEC 60870 i 61850) oraz sterownik PLC1 z firmy Bart-Com [38] ze zdalnymi modułami wejścia/wyjścia (Profibus DP, Modbus RTU) dedykowany do inteligentnych domów.

8. Podsumowanie

W artykule zaprezentowano przenośne środowisko inżynierskie, przeznaczone do implementacji w sterownikach przemysłowych z 8-, 16- lub 32-bitowymi procesorami. Środowisko oparte jest na maszynie wirtualnej wykonującej kod pośredni VMASM, do którego kompilowane są programy z języków ST lub IL. Języki graficzne są najpierw tłumaczone do ST. Kompilator i maszyna wirtualna są parametryzowane za pomocą plików konfiguracyjnych XML. Prototypy niskopoziomowych funkcji wbudowane w maszynę wirtualną umożliwiają jej wdrożenie na różnych platformach sprzętowych. Dzięki rozdzieleniu oprogramowania HMI na warstwy zależne i niezależne od sprzętu, projekty HMI mogą być wyświetlane na różnych urządzeniach. Generowany przez kompilator plik XML z danymi o zmiennych globalnych umożliwia ich powiązanie z interfejsami komunikacyjnymi i I/O. Zazwyczaj producenci urządzeń automatyki mogą samodzielnie zaimplementować opisane środowisko inżynierskie.

Rozszerzenia środowiska o programowanie zorientowane obiektowo, wykonywanie zadań wyzwalane zdarzeniowo i przeniesienie wieloprojektowego środowiska na wielordzeniowe procesory, ze współbieżnym uruchamianiem projektów na poszczególnych rdzeniach, są obecnie opracowywane.

Podziękowania

Autorzy dziękują Marcinowi Jamro za pomoc przy rozwoju środowiska kilka lat temu.

Projekt finansowany w ramach programu Ministra Nauki i Szkolnictwa Wyższego pod nazwą „Regionalna Inicjatywa Doskonałości" w latach 2019–2022 nr projektu 027/RID/2018/19 całkowita kwota finansowania 11 999 900 zł.

Bibliografia

1. Vogel-Heuser B., Schütz D., Frank T., Legat C., *Model-driven engineering of Manufacturing Automation Software Projects – A SysMLbased approach*, "Mechatronics", Vol. 24, No. 7, 2014, 883–897, DOI: 10.1016/j.mechatronics.2014.05.003.
2. Thramboulidis K., *A cyber–physical system-based approach for industrial automation systems*, "Computers in Industry", Vol. 72, 2015, 92–102, DOI: 10.1016/j.compind.2015.04.006.
3. IEC 61131-3 - Programmable controllers - Part 3: Programming languages, 2003, 2013.
4. Hsieh S.-J., Cheng Y.-T., *Algorithm and intelligent tutoring system design for programmable controller programming*, "The International Journal of Advanced Manufacturing Technology", Vol. 71, No. 5–8, 2014, 1099–1115, DOI: 10.1007/s00170-013-5539-z.
5. *ISaGRAF Technology*, [www.isagraf.com].
6. COPA-DATA France, *Software components for industrial control*, [www.straton-plc.com].
7. Rzońca D., Sadolewski J., Stec A., Świder Z., Trybus B., Trybus L., *Mini-DCS system programming in IEC 61131-3 Structured Text*, "Journal of Automation, Mobile Robotics and Intelligent Systems", Vol. 2, No. 3, 2008, 48–54.
8. Cisek J., Mikluszka W., Świder Z., Trybus L., *A Low-Cost DCS with Multifunction Instruments and CAN Bus*, "IFAC Proceedings Volumes", Vol. 34, No. 29, 2001, 64–69, DOI: 10.1016/S1474-6670(17)32794-5.
9. Hajduk Z., Trybus B., Sadolewski J., *Architecture of FPGA Embedded Multiprocessor Programmable Controller*, "IEEE Transactions on Industrial Electronics", Vol. 62, No. 5, 2015, 2952–2961, DOI: 10.1109/TIE.2014.2362888.
10. Jamro M., Rzońca D., Rząsa W., *Testing communication tasks in distributed control systems with SysML and Timed Colored Petri Nets model*, "Computers in Industry", Vol. 71, 2015, 77–87, DOI: 10.1016/j.compind.2015.03.007.
11. Jamro M., *POU-Oriented Unit Testing of IEC 61131-3 Control Software*, "IEEE Transactions on Industrial Informatics", Vol. 11, No. 5, 2015, 1119–1129, DOI: 10.1109/TII.2015.2469257.
12. B. Venners, *Inside the Java Virtual Machine*. New York: McGraw-Hill Inc., 1997.
13. Richter J., Richter A., *CLR via C#*. Microsoft Press, 2006.
14. Schoeberl M., *A Java Processor Architecture for Embedded Real-time Systems*, "Journal of Systems Architecture", Vol. 54, No. 1–2, 2008, 265–286, DOI: 10.1016/j.sysarc.2007.06.001.
15. Zhou C., Chen H., *Development of a PLC Virtual Machine Orienting IEC 61131-3 Standard*, [in:] 2009 International Conference on Measuring Technology and Mechatronics Automation, Vol. 3, 2009, 374–379, DOI: 10.1109/ICMTMA.2009.422.
16. Zhang M., Lu Y., Xia T., *The Design and Implementation of Virtual Machine System in Embedded SoftPLC System*, [in:] 2013 International Conference on Computer Sciences and Applications, 775–778, DOI: 10.1109/CSA.2013.185.
17. Cavalieri S., Puglisi G., Scroppo M.S., Galvagno L., Moving IEC 61131-3 applications to a computing framework based on CLR Virtual Machine, [in:] 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), 1–8, DOI: 10.1109/ETFA.2016.7733632.
18. De Tommasi G., Pironti A., *An educational open-source tool for the design of IEC 61131-3 compliant automation software*, [in:] 2008 International Symposium on Power Electronics, Electrical Drives, Automation and Motion, 486–491, DOI: 10.1109/SPEEDHAM.2008.4581144.
19. Shin S., Kwon M., Rho S., *Whimori CDK: A Control Program Development Kit*, [in:] 2009 International Conference on Computing, Engineering and Information, 115–118, DOI: 10.1109/ICC.2009.33.
20. Palma L.B., Brito V., Rosas J., Gil P., *WEB PLC simulator for ST programming*, [in:] 2017 4th Experiment@International Conference (exp.at'17), 303–308, DOI: 10.1109/EXPAT.2017.7984410.
21. Inzunza Villagómez H.I., Pérez Arce B., Hernández Ruiz S.I., López Corella J.A., *Design and implementation of a development environment on ladder diagram (HT-PLC) for Arduino with Ethernet connection*, [in:] 2018 IEEE International Conference on Automation/XXIII Congress of the Chilean Association of Automatic Control (ICA-ACCA), 1–6, DOI: 10.1109/ICA-ACCA.2018.8609850.
22. Cooper K.D., Torczon L., *Engineering a Compiler* (Second Edition). Morgan Kaufmann, Boston 2012.
23. Fen G., Ning W., *A Transformation Algorithm of Ladder Diagram into Instruction List Based on AOV Digraph and Binary Tree*, [in:] TENCON 2006 – 2006 IEEE Region 10 Conference, 1–4, DOI: 10.1109/TENCON.2006.343937.
24. Huang L., Liu W., Liu Z., *Algorithm of transformation from PLC ladder diagram to structured text*, [in:] 2009 9th International Conference on Electronic Measurement Instruments, 4-778–4-782, DOI: 10.1109/ICEMI.2009.5274701.
25. Milik A., Hrynkiewicz E., *Distributed PLC Based on Multicore CPUs – Architecture and Programming*, "IFAC-PapersOnLine", Vol. 49, No. 25, 2016, 1–7, DOI: 10.1016/j.ifacol.2016.12.001.
26. Fiset J.-Y., *Human-Machine Interface Design for Process Control Applications*. Instrumentation, Systems and Automation Society, 2009.

27. VDI/VDE 3699 Process control using display screens, 2015.
28. Jamro M., Rzońca D., *Automatic connections in IEC 61131-3 Function Block Diagrams*, [in:] 2013 Federated Conference on Computer Science and Information Systems, 463–469.
29. Hart P.E., Nilsson N.J., Raphael B., *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, "IEEE Transactions on Systems Science and Cybernetics", Vol. 4, No. 2, 1968, 100–107, DOI: 10.1109/TSSC.1968.300136.
30. OMG, "OMG Systems Modeling Language, V1.3," 2012.
31. Jamro M., Rzońca D., *Agile and hierarchical round-trip engineering of IEC 61131-3 control software*, "Computers in Industry", Vol. 96, 2018, 1–9, DOI: 10.1016/j.compind.2018.01.004.
32. Sadolewski J., Szmyd E., *Polski kompilator oraz sterowniki programowane za pomocą języka IL normy IEC 61131-3*, „Pomiary Automatyka Robotyka", R. 13, Nr 2, 2009, 615–622.
33. Lumel S.A. [www.lumel.com.pl/en].
34. Light OPC Server. [www.ipi.ac.ru].
35. Praxis Automation Technology B.V. [www.praxis-automation.nl].
36. Nauka i Technika. [www.nit.pl].
37. iGrid T&D. [www.igrid-td.com].
38. Bart-Com. [www.bart-com.pl].

Implementation of Engineering Environment – Case Study: CPDev Package

Abstract: The IEC 61131-3 standard defines five languages, dedicated for programming industrial controllers. The standard is commonly used, there are numerous engineering environments fully or partially compatible with it. Several academic solutions have been described in the literature, but they typically implement only selected parts of the IEC 61131-3 standard (e.g. only one or two languages). On the other hand, commercial engineering environments usually implement all languages, but their documentation focuses on the application of the environment, whereas details about internal architecture and implementation are rarely disclosed. The paper describes such internal details of the CPDev engineering environment. The architecture based on the virtual machine makes the environment portable, thus facilitate implementation on diverse hardware platforms. Several industrial implementations of CPDev are also mentioned.

Keywords: IEC 61131-3, engineering environment, intermediate language, virtual machine, translator, HMI, communication, industrial controller

dr inż. Dariusz Rzońca
drzonca@kia.prz.edu.pl
ORCID: 0000-0001-5724-0978

Licencjat matematyki (Uniwersytet Rzeszowski 2002), magister inżynier informatyki (Politechnika Rzeszowska 2004), doktor nauk technicznych w dyscyplinie informatyka, specjalność przemysłowe systemy informatyki (Politechnika Śląska 2012). Od 2004 roku asystent, a od 2013 adiunkt w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej. Jego zainteresowania naukowe koncentrują się na kolorowanych sieciach Petriego oraz zagadnieniach związanych z komunikacją w systemach automatyki.

dr inż. Jan Sadolewski
js@kia.prz.edu.pl
ORCID: 0000-0001-7370-9027

Absolwent Wydziału Elektrotechniki i Informatyki Politechniki Rzeszowskiej (2006). W 2012 roku uzyskał stopień doktora nauk technicznych w dyscyplinie informatyka na Wydziale Automatyki, Elektroniki i Informatyki Politechniki Śląskiej w Gliwicach. Jego zainteresowania naukowe koncentrują się wokół języków programowania, tworzenia kompilatorów oraz środowisk wykonawczych.

dr inż. Andrzej Stec
astec@kia.prz.edu.pl
ORCID: 0000-0002-5615-3626

Absolwent Wydziału Elektrotechniki i Informatyki Politechniki Rzeszowskiej (1993). Stopień doktora nauk technicznych uzyskał w 2004 r. Obecnie adiunkt dydaktyczny w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej. Zajmuje się zagadnieniami dotyczącymi systemów wbudowanych i rozproszonych systemów sterowania.

dr hab. inż. Zbigniew Świder
swiderzb@prz.edu.pl
ORCID: 0000-0003-3504-5340

Absolwent Wydziału Elektrotechniki i Informatyki Politechniki Rzeszowskiej (1984). Na Wydziale Automatyki, Elektroniki i Informatyki Politechniki Śląskiej w Gliwicach uzyskał stopień doktora nauk technicznych (1992) oraz doktora habilitowanego nauk technicznych (2004). Od początku pracy zawodowej zajmuje się sterownikami mikroprocesorowymi, a ostatnio metodami samostrojenia i adaptacji nastaw regulatorów, komputerowymi systemami automatyki oraz środowiskami inżynierskimi do programowania przemysłowych układów sterowania.

dr inż. Bartosz Trybus
btrybus@kia.prz.edu.pl
ORCID: 0000-0002-4588-3973

Adiunkt w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej. Ukończył studia na Wydziale Elektrycznym, Automatyki, Informatyki i Elektroniki AGH w Krakowie. Doktorat z informatyki uzyskał w 2004 r. Jego główne badania dotyczą systemów czasu rzeczywistego i środowisk wykonawczych oprogramowania sterującego.

prof. dr hab. inż. Leszek Trybus
ltrybus@prz-rzeszow.pl
ORCID: 0000-0002-1415-3679

Ukończył AGH Kraków (1970), gdzie również uzyskał doktorat, habilitację i tytuł. Od ukończenia studiów pracuje w Politechnice Rzeszowskiej. Autor ponad 200 publikacji. Wieloletni członek Komitetu Automatyki i Robotyki PAN. Zajmuje się regulatorami mikroprocesorowymi, rozproszonymi systemami.